Network Working Group
Request for Comments: 1320
Obsoletes: RFC 1186

R. Rivest
MIT Laboratory for Computer Science
and RSA Data Security, Inc.

April 1992

Алгоритм MD4

The MD4 Message-Digest Algorithm

Статус документа

Этот документ содержит информацию для сообщества Internet. Документ не задаёт каких-либо стандартов. Разрешается свободное распространение документа.

Благодарности

Автор благодарит Don Coppersmith, Burt Kaliski, Ralph Merkle и Noam Nisan за множество полезных комментариев и предложений.

Оглавление

1. Введение

В этом документе описывается алгоритм цифровых подписей[1] MD4. Этот алгоритм принимает на входе сообщение произвольного размера и выдает в результате 128-битовый «отпечаток» (fingerprint) или цифровую подпись (message digest). Предполагается, что потребуется нереальный объем вычислений для создания двух сообщений, цифровые подписи которых совпадут, или подбора сообщения по имеющейся цифровой подписи. Алгоритм MD4 предназначен для создания цифровой подписи (сигнатуры), когда требуется безопасно «сжать» большой файл перед тем, как шифровать его с использованием закрытого (секретного) ключа в системах с открытыми ключами типа RSA.

Алгоритм MD4 разработан для обеспечения достаточной скорости на 32-битовых машинах. В дополнение к этому MD4 не требует использования больших таблиц подстановки; кодирование алгоритма может быть достаточно компактным.

Алгоритм MD4 открыт и возможно будет адаптирован в качестве стандартного.

Данный документ заменяет выпущенный в октябре 1990 документ RFC 1186 [2]. Основное отличие между документами заключается в том, что представленный здесь вариант реализации MD4 является более переносимым.

Для приложений OSI идентификатор объекта MD4 имеет форму

```
md4 OBJECT IDENTIFIER ::= {iso(1) member-body(2) US(840) rsadsi(113549) digestAlgorithm(2) 4}
```

В идентификаторе типа X.509 (AlgorithmIdentifier [3]) параметры для MD4 должны иметь тип NULL.

2. Термины и обозначения

В этом документе термин «слово» (word) используется для 32-битовых значений, а «байт» - для восьмибитовых. Последовательности битов могут интерпретироваться естественным путем, как последовательность байтов, в которой каждая группа из 8 последовательных битов интерпретируется как байт, где старший (наиболее значимый) бит располагается первым. Подобно этому последовательность байтов может интерпретироваться как последовательность 32-битовых слов, в которой каждая группа из 4 последовательных байтов интерпретируется как слово, где младший (наименее значимый) указывается первым.

[1] Message-digest algorithm.

Запись **x_i** означает x i-ое (x[i]). Если индекс представляет собой выражение, оно будет заключаться в фигурные скобки x_{i+1}. Символ ^ используется для обозначения степени (экспонента) - x^i означает x в степени i.

Символ "+" обозначает сложение слов (т. е., сложение с использованием модуля 2^{32}). Запись **X <<< s** означает 32-битовое слово, полученное циклическим сдвигом X влево на s позиций. **not(X)** означает побитовое дополнение X, а **X v Y** побитовую операцию **X OR Y** (X ИЛИ Y). Запись **X xor Y** означает побитовую операцию **X XOR Y** (исключающее ИЛИ), а **XY** побитовую операцию **X AND Y** (X И Y).

3. Описание алгоритма MD4

Начнем с допущения о наличии на входе сообщения размером b битов, для которого нужно создать цифровую подпись. Здесь b означает неотрицательное целое число, b может принимать нулевое значение и не обязано быть кратным 8. Это значение может быть неограниченно большим. Биты исходного сообщения будем представлять следующим образом:

```
m_0 m_1 ... m_{b-1}
```

Для создания цифровой подписи выполняется процесс из 5 описанных ниже этапов.

3.1 Этап 1. Добавление битов заполнения

Сообщение дополняется до размера (в битах), конгруэнтного 448 по модулю 512. Т. е., размер сообщения устанавливается таким, чтобы после добавления 64 битов размер был кратен 512. Заполнение производится во всех случаях, даже если размер исходного сообщения конгруэнтен 448 по модулю 512.

Заполнение происходит следующим образом: сначала в конце сообщения добавляется один бит, имеющий значение 1, а потом добавляются биты, имеющие значение 0, до размера, конгруэнтного 448 по модулю 512. В любом случае число добавляемых битов не может быть меньше 1 м больше 512.

3.2 Этап 2. Добавление размера сообщения

64-битовое представление b (размер исходного сообщения в битах) добавляется в конец результата предыдущего этапа. В редких случаях, когда b больше 2^{64}, используются только младшие 64 бита значения b. Биты добавляются в конце, как два 32-битовых слова, в которых младший байт размещается в начале, как было условлено выше.

После этого размер полученного сообщения (исходное сообщение + заполнение + размер) будет кратен 512 битам. Следовательно, результирующее сообщение будет содержать целое число блоков по 16 слов (32 бита в каждом слове). Пусть M[0 ... N-1] обозначает слова полученного в результате сообщения (N кратно 16).

3.3 Этап 3. Инициализация буфера MD

Для расчета цифровой подписи используется буфер на 4 слова (A,B,C,D). Каждое из слов A, B, C, D представляет собой 32-битовый регистр. Эти регистры инициализируются приведенными ниже значениями (шестнадцатеричное представление, сначала младший байт):

```
слово A: 01 23 45 67
слово B: 89 ab cd ef
слово C: fe dc ba 98
слово D: 76 54 32 10
```

3.4 Этап 4. Обработка сообщения блоками по 16 слов

Сначала определим три дополнительных функции, каждая из которых принимает три 32-битовых слова в качестве аргументов и возвращает одно 32-битовое слово.

```
F(X,Y,Z) = XY v not(X) Z
G(X,Y,Z) = XY v XZ v YZ
H(X,Y,Z) = X xor Y xor Z
```

Для каждого бита функция F работает как условие – если X, то Y, иначе Z. Функцию F можно определить с использованием сложения (+) вместо операции ИЛИ (v), поскольку XY и not(X)Z никогда не будут давать 1 в одной битовой позиции. Для каждой битовой позиции G работает как мажоритарная функция - если хотя бы один из битов X, Y, Z имеет значение 1, G будет устанавливать значение 1 для соответствующего бита результата, в противном случае G будет устанавливать 0. Интересно отметить, что при условии, когда биты X, Y и Z независимы и не образованы смещением (unbiased), каждый бит функции G(X,Y,Z) будет независимым и не будет образован смещением. Функция H представляет собой побитовую операцию «исключающее ИЛИ» (xor) или функцию «четности» (parity) переданных ей аргументов; свойства этой функции подобны свойствам F и G.

Ниже показана обработка блоков сообщения.

```
/* обработать каждый блок из 16 слов. */
For i = 0 to N/16-1 do

        /* скопировать блок i в X. */
        For j = 0 to 15 do
                Set X[j] to M[i*16+j].
        end /* цикл по j */

        /* сохранить A как AA, B как BB, C как CC, D как DD. */
        AA = A
        BB = B
        CC = C
        DD = D

        /* Раунд 1. */
        /* [abcd k s] обозначает операцию a = (a + F(b,c,d) + X[k]) <<< s. */
        /* Выполним следующие 16 операций. */
```

```
        [ABCD  0  3]  [DABC  1  7]  [CDAB  2 11]  [BCDA  3 19]
        [ABCD  4  3]  [DABC  5  7]  [CDAB  6 11]  [BCDA  7 19]
        [ABCD  8  3]  [DABC  9  7]  [CDAB 10 11]  [BCDA 11 19]
        [ABCD 12  3]  [DABC 13  7]  [CDAB 14 11]  [BCDA 15 19]

        /* Раунд 2. */
        /* [abcd k s] обозначает операцию a = (a + G(b,c,d) + X[k] + 5A827999) <<< s). */
        /* Выполним следующие 16 операций. */
        [ABCD  0  3]  [DABC  4  5]  [CDAB  8  9]  [BCDA 12 13]
        [ABCD  1  3]  [DABC  5  5]  [CDAB  9  9]  [BCDA 13 13]
        [ABCD  2  3]  [DABC  6  5]  [CDAB 10  9]  [BCDA 14 13]
        [ABCD  3  3]  [DABC  7  5]  [CDAB 11  9]  [BCDA 15 13]

        /* Раунд 3. */
        /* [abcd k s] обозначает операцию a = (a + H(b,c,d) + X[k] + 6ED9EBA1) <<< s). */
        /* Выполним следующие 16 операций. */
        [ABCD  0  3]  [DABC  8  9]  [CDAB  4 11]  [BCDA 12 15]
        [ABCD  2  3]  [DABC 10  9]  [CDAB  6 11]  [BCDA 14 15]
        [ABCD  1  3]  [DABC  9  9]  [CDAB  5 11]  [BCDA 13 15]
        [ABCD  3  3]  [DABC 11  9]  [CDAB  7 11]  [BCDA 15 15]

        /* После этого добавим к каждому из 4 регистров значение, которое этот регистр имел
до начала выполнения этого блока. */
        A = A + AA
        B = B + BB
        C = C + CC
        D = D + DD
end /* цикл по i */
```

Примечание. Значение 5A..99 является шестнадцатеричной 32-битовой константой, записанной начиная со старшего бита. Эта константа представляет собой квадратный корень из 2. Восьмеричное значение этой константы равно 013240474631.

Значение 6E..A1 является шестнадцатеричной 32-битовой константой, записанной начиная со старшего бита. Эта константа представляет собой квадратный корень из 3. Восьмеричное значение этой константы равно 015666365641.

См. книгу Knuth, The Art of Programming, Volume 2 (Seminumerical Algorithms), Second Edition (1981), Addison-Wesley[1]. Table 2, стр. 660.

3.5 Этап 5. Вывод

Сигнатура сообщения выводится как A, B, C, D (т. е., мы начинаем вывод с младшего байта A и закачиваем старшим байтом D).

На этом описание алгоритма MD4 завершается. Пример реализации на языке C дан в приложении.

4. Заключение

Алгоритм цифровых подписей MD4 прост в реализации и создает «отпечатки» (fingerprint) или цифровые подписи для сообщений произвольной длины. Предполагается, что для создания двух сообщений с одинаковыми сигнатурами потребуется порядка 2^{64} операций, а для подбора сообщения по имеющейся сигнатуре - порядка 2^{128} операций. Алгоритм MD4 был тщательно исследован на предмет поиска слабых мест. Однако алгоритм является достаточно новым и требуется дополнительный анализ, как и для всех новинок этого типа.

Литература

[1] Rivest, R., "The MD4 message digest algorithm", in A.J. Menezes and S.A. Vanstone, editors, Advances in Cryptology - CRYPTO '90 Proceedings, pages 303-311, Springer-Verlag, 1991.

[2] Rivest, R., "The MD4 Message Digest Algorithm", RFC 1186, MIT, October 1990.

[3] CCITT Recommendation X.509 (1988), "The Directory - Authentication Framework".

[4] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, MIT and RSA Data Security, Inc, April 1992.

Приложение A – Пример реализации

В это приложение включены файлы:

```
global.h – общий файл заголовков
md4.h – файл заголовков для MD4
md4c.c – исходный код реализации MD4
mddriver.c – тестовый драфвер для MD2, MD4 и MD5
```

Драйвер по умолчанию компилируется для проверки MD5 и может быть скомпилирован для проверки MD2 или MD4, если в командной строке компилятора C задать для MD значение 2 или 4, соответственно.

Реализация переносима и должна работать на разных платформах. Однако ее несложно оптимизировать для конкретной платформы (оставим это в качестве упражнения для читателей). Например, для платформ типа «little-endian», где байт с наименьшим адресом в 32-битовом слове является наименее значимым и нет ограничений по выравниванию, вызов Decode в MD4Transform можно заменить преобразованием типа.

A.1 global.h

```
/* GLOBAL.H – типы и константы RSAREF */
```

[1]Эта книга неоднократно издавалась на русском языке. *Прим. перев.*

```
/* Для PROTOTYPES следует использовать значение 1 тогда и только тогда, когда компилятор
поддерживает прототипы аргументов в функциях. Приведенный ниже код устанавливает для PROTOTYPES по
умолчанию значение 0, если этот параметр уже не задан флагами компилятора C.
 */
#ifndef PROTOTYPES
#define PROTOTYPES 0
#endif

/* POINTER определяет базовый тип указателя */
typedef unsigned char *POINTER;

/* UINT2 определяет 2-байтовое слово */
typedef unsigned short int UINT2;

/* UINT4 определяет 4-байтовое слово */
typedef unsigned long int UINT4;

/* PROTO_LIST определяется в зависимости от определенного ранее значения PROTOTYPES.
При использовании PROTOTYPES список PROTO_LIST будет возвращать прототипы, иначе будет пустым.
 */
#if PROTOTYPES
#define PROTO_LIST(list) list
#else
#define PROTO_LIST(list) ()
#endif
```

A.2 md4.h

```
/* MD4.H - заголовочный файл для MD4C.C */

/* Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All
   rights reserved.

   License to copy and use this software is granted provided that it
   is identified as the "RSA Data Security, Inc. MD4 Message-Digest
   Algorithm" in all material mentioning or referencing this software
   or this function.

   License is also granted to make and use derivative works provided
   that such works are identified as "derived from the RSA Data
   Security, Inc. MD4 Message-Digest Algorithm" in all material
   mentioning or referencing the derived work.

   RSA Data Security, Inc. makes no representations concerning either
   the merchantability of this software or the suitability of this
   software for any particular purpose. It is provided "as is"
   without express or implied warranty of any kind.

   These notices must be retained in any copies of any part of this
   documentation and/or software.
 */

/* Контекст MD4. */
typedef struct {
       UINT4 state[4];             /* состояние (ABCD) */
       UINT4 count[2];               /* число битов, modulo 2^64 (сначала младший) */
       unsigned char buffer[64];  /* входной буфер*/
} MD4_CTX;

void MD4Init PROTO_LIST ((MD4_CTX *));
void MD4Update PROTO_LIST ((MD4_CTX *, unsigned char *, unsigned int));
void MD4Final PROTO_LIST ((unsigned char [16], MD4_CTX *));
```

A.3 md4c.c

```
/* MD4C.C - RSA Data Security, Inc., алгоритм цифровой подписи MD4 */

/* Copyright (C) 1990-2, RSA Data Security, Inc. All rights reserved.

   License to copy and use this software is granted provided that it
   is identified as the "RSA Data Security, Inc. MD4 Message-Digest
   Algorithm" in all material mentioning or referencing this software
   or this function.

   License is also granted to make and use derivative works provided
   that such works are identified as "derived from the RSA Data
   Security, Inc. MD4 Message-Digest Algorithm" in all material
   mentioning or referencing the derived work.

   RSA Data Security, Inc. makes no representations concerning either
   the merchantability of this software or the suitability of this
   software for any particular purpose. It is provided "as is"
   without express or implied warranty of any kind.

   These notices must be retained in any copies of any part of this
   documentation and/or software.
 */
```

```
#include "global.h"
#include "md4.h"

/* Константы для подпрограммы MD4Transform. */
#define S11 3
#define S12 7
#define S13 11
#define S14 19
#define S21 3
#define S22 5
#define S23 9
#define S24 13
#define S31 3
#define S32 9
#define S33 11
#define S34 15

static void MD4Transform PROTO_LIST ((UINT4 [4], unsigned char [64]));
static void Encode PROTO_LIST ((unsigned char *, UINT4 *, unsigned int));
static void Decode PROTO_LIST ((UINT4 *, unsigned char *, unsigned int));
static void MD4_memcpy PROTO_LIST ((POINTER, POINTER, unsigned int));
static void MD4_memset PROTO_LIST ((POINTER, int, unsigned int));

static unsigned char PADDING[64] = {
        0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

/* F, G и H являются базовыми функциями MD4. */
#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))
#define G(x, y, z) (((x) & (y)) | ((x) & (z)) | ((y) & (z)))
#define H(x, y, z) ((x) ^ (y) ^ (z))

/* ROTATE_LEFT циклически смещает x влево на n битов. */
#define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))

/* Преобразования FF, GG и HH для раундов 1, 2, 3.
Циклический сдвиг отделен от добавления для предотвращения повторного расчета. */
#define FF(a, b, c, d, x, s) { \
    (a) += F ((b), (c), (d)) + (x); \
    (a) = ROTATE_LEFT ((a), (s)); \
  }
#define GG(a, b, c, d, x, s) { \
    (a) += G ((b), (c), (d)) + (x) + (UINT4)0x5a827999; \
    (a) = ROTATE_LEFT ((a), (s)); \
  }
#define HH(a, b, c, d, x, s) { \
    (a) += H ((b), (c), (d)) + (x) + (UINT4)0x6ed9eba1; \
    (a) = ROTATE_LEFT ((a), (s)); \
  }

/* Инициализация MD4. Начинает работу MD4, записывая новый контекст. */
void MD4Init (context)
MD4_CTX *context;                                        /* контекст */
{
        context->count[0] = context->count[1] = 0;

        /* загрузка "магических" констант инициализации. */
        context->state[0] = 0x67452301;
        context->state[1] = 0xefcdab89;
        context->state[2] = 0x98badcfe;
        context->state[3] = 0x10325476;
}

/* MD4 блокирует операцию обновления. Продолжается создание сигнатуры MD4, обработка другого
  блока сообщения и обновление контекста.
 */
void MD4Update (context, input, inputLen)
MD4_CTX *context;                        /* контекст */
unsigned char *input;                    /* входной блок */
unsigned int inputLen;                   /* размер входного блока */
{
        unsigned int i, index, partLen;

        /* Расчет числа байтов mod 64 */
        index = (unsigned int)((context->count[0] >> 3) & 0x3F);
        /* Обновление числа битов */
        if ((context->count[0] += ((UINT4)inputLen << 3)) < ((UINT4)inputLen << 3))
                context->count[1]++;
        context->count[1] += ((UINT4)inputLen >> 29);

        partLen = 64 - index;
```

```
        /* Преобразование возможное число раз. */
        if (inputLen >= partLen) {
                MD4_memcpy ((POINTER)&context->buffer[index], (POINTER)input, partLen);
                MD4Transform (context->state, context->buffer);

                for (i = partLen; i + 63 < inputLen; i += 64)
                        MD4Transform (context->state, &input[i]);

                index = 0;
        } else
                i = 0;

        /* Буферизация оставшихся входных данных */
        MD4_memcpy ((POINTER)&context->buffer[index], (POINTER)&input[i], inputLen-i);
}

/* Завершение работы MD4. Окончание расчета цифровой подписи MD4, вывод сигнатуры и обнуление
   контекста.
 */
void MD4Final (digest, context)
unsigned char digest[16];                       /* цифровая подпись */
MD4_CTX *context;                               /* контекст */
{
        unsigned char bits[8];
        unsigned int index, padLen;

        /* Сохранение числа битов */
        Encode (bits, context->count, 8);

        /* Заполнение до 56 mod 64. */
        index = (unsigned int)((context->count[0] >> 3) & 0x3f);
        padLen = (index < 56) ? (56 - index) : (120 - index);
        MD4Update (context, PADDING, padLen);

        /* Добавление размера (до заполнения) */
        MD4Update (context, bits, 8);
        /* Сохранение состояния в дайджесте */
        Encode (digest, context->state, 16);

        /* Обнуление чувствительной информации. */
        MD4_memset ((POINTER)context, 0, sizeof (*context));
}

/* Базовое преобразование MD4. Преобразует состояние поблочно. */
static void MD4Transform (state, block)
UINT4 state[4];
unsigned char block[64];
{
        UINT4 a = state[0], b = state[1], c = state[2], d = state[3], x[16];

        Decode (x, block, 64);

        /* Раунд 1 */
        FF (a, b, c, d, x[ 0], S11); /* 1 */
        FF (d, a, b, c, x[ 1], S12); /* 2 */
        FF (c, d, a, b, x[ 2], S13); /* 3 */
        FF (b, c, d, a, x[ 3], S14); /* 4 */
        FF (a, b, c, d, x[ 4], S11); /* 5 */
        FF (d, a, b, c, x[ 5], S12); /* 6 */
        FF (c, d, a, b, x[ 6], S13); /* 7 */
        FF (b, c, d, a, x[ 7], S14); /* 8 */
        FF (a, b, c, d, x[ 8], S11); /* 9 */
        FF (d, a, b, c, x[ 9], S12); /* 10 */
        FF (c, d, a, b, x[10], S13); /* 11 */
        FF (b, c, d, a, x[11], S14); /* 12 */
        FF (a, b, c, d, x[12], S11); /* 13 */
        FF (d, a, b, c, x[13], S12); /* 14 */
        FF (c, d, a, b, x[14], S13); /* 15 */
        FF (b, c, d, a, x[15], S14); /* 16 */

        /* Раунд 2 */
        GG (a, b, c, d, x[ 0], S21); /* 17 */
        GG (d, a, b, c, x[ 4], S22); /* 18 */
        GG (c, d, a, b, x[ 8], S23); /* 19 */
        GG (b, c, d, a, x[12], S24); /* 20 */
        GG (a, b, c, d, x[ 1], S21); /* 21 */
        GG (d, a, b, c, x[ 5], S22); /* 22 */
        GG (c, d, a, b, x[ 9], S23); /* 23 */
        GG (b, c, d, a, x[13], S24); /* 24 */
        GG (a, b, c, d, x[ 2], S21); /* 25 */
        GG (d, a, b, c, x[ 6], S22); /* 26 */
        GG (c, d, a, b, x[10], S23); /* 27 */
        GG (b, c, d, a, x[14], S24); /* 28 */
        GG (a, b, c, d, x[ 3], S21); /* 29 */
        GG (d, a, b, c, x[ 7], S22); /* 30 */
        GG (c, d, a, b, x[11], S23); /* 31 */
```

```
        GG (b, c, d, a, x[15], S24); /* 32 */

        /* Раунд 3 */
        HH (a, b, c, d, x[ 0], S31); /* 33 */
        HH (d, a, b, c, x[ 8], S32); /* 34 */
        HH (c, d, a, b, x[ 4], S33); /* 35 */
        HH (b, c, d, a, x[12], S34); /* 36 */
        HH (a, b, c, d, x[ 2], S31); /* 37 */
        HH (d, a, b, c, x[10], S32); /* 38 */
        HH (c, d, a, b, x[ 6], S33); /* 39 */
        HH (b, c, d, a, x[14], S34); /* 40 */
        HH (a, b, c, d, x[ 1], S31); /* 41 */
        HH (d, a, b, c, x[ 9], S32); /* 42 */
        HH (c, d, a, b, x[ 5], S33); /* 43 */
        HH (b, c, d, a, x[13], S34); /* 44 */
        HH (a, b, c, d, x[ 3], S31); /* 45 */
        HH (d, a, b, c, x[11], S32); /* 46 */
        HH (c, d, a, b, x[ 7], S33); /* 47 */
        HH (b, c, d, a, x[15], S34); /* 48 */

        state[0] += a;
        state[1] += b;
        state[2] += c;
        state[3] += d;

        /* Обнуление чувствительных данных. */
        MD4_memset ((POINTER)x, 0, sizeof (x));
}

/* Кодирование входных данных (UINT4) в выходные (unsigned char).
   Предполагается, что размер кратен 4.
 */
static void Encode (output, input, len)
unsigned char *output;
UINT4 *input;
unsigned int len;
{
        unsigned int i, j;

        for (i = 0, j = 0; j < len; i++, j += 4) {
                output[j] = (unsigned char)(input[i] & 0xff);
                output[j+1] = (unsigned char)((input[i] >> 8) & 0xff);
                output[j+2] = (unsigned char)((input[i] >> 16) & 0xff);
                output[j+3] = (unsigned char)((input[i] >> 24) & 0xff);
        }
}

/* Декодирование входных данных (unsigned char) в выходные (UINT4).
   Предполагается, что размер кратен 4.
 */
static void Decode (output, input, len)

UINT4 *output;
unsigned char *input;
unsigned int len;
{
        unsigned int i, j;

        for (i = 0, j = 0; j < len; i++, j += 4)
        output[i] = ((UINT4)input[j]) | (((UINT4)input[j+1]) << 8) |
                    (((UINT4)input[j+2]) << 16) | (((UINT4)input[j+3]) << 24);
}

/* Примечание: По возможности замените цикл for стандартной функцией memcpy. */
static void MD4_memcpy (output, input, len)
POINTER output;
POINTER input;
unsigned int len;
{
        unsigned int i;

        for (i = 0; i < len; i++)
                output[i] = input[i];
}

/* Примечание: По возможности замените цикл for стандартной функцией memset. */
static void MD4_memset (output, value, len)
POINTER output;
int value;
unsigned int len;
{
        unsigned int i;

        for (i = 0; i < len; i++)
                ((char *)output)[i] = (char)value;
}
```

A.4 mddriver.c

```
/* MDDRIVER.C - тест-драйвер для MD2, MD4 и MD5 */

/* Copyright (C) 1990-2, RSA Data Security, Inc. Created 1990. All
   rights reserved.

   RSA Data Security, Inc. makes no representations concerning either
   the merchantability of this software or the suitability of this
   software for any particular purpose. It is provided "as is"
   without express or implied warranty of any kind.

   These notices must be retained in any copies of any part of this
   documentation and/or software.

 */

/* Включенное ниже определение по умолчанию обеспечивает проверку MD5, если флаги компилятора
   C не задают иного.
 */
#ifndef MD
#define MD 5
#endif

#include <stdio.h>
#include <time.h>
#include <string.h>
#include "global.h"
#if MD == 2
#include "md2.h"
#endif
#if MD == 4
#include "md4.h"
#endif
#if MD == 5
#include "md5.h"
#endif

/* Размер и число тестовых блоков. */
#define TEST_BLOCK_LEN 1000
#define TEST_BLOCK_COUNT 1000

static void MDString PROTO_LIST ((char *));
static void MDTimeTrial PROTO_LIST ((void));
static void MDTestSuite PROTO_LIST ((void));
static void MDFile PROTO_LIST ((char *));
static void MDFilter PROTO_LIST ((void));
static void MDPrint PROTO_LIST ((unsigned char [16]));

#if MD == 2
#define MD_CTX MD2_CTX
#define MDInit MD2Init
#define MDUpdate MD2Update
#define MDFinal MD2Final
#endif
#if MD == 4
#define MD_CTX MD4_CTX
#define MDInit MD4Init
#define MDUpdate MD4Update
#define MDFinal MD4Final
#endif
#if MD == 5
#define MD_CTX MD5_CTX
#define MDInit MD5Init
#define MDUpdate MD5Update
#define MDFinal MD5Final
#endif


/* Драйвер.
Аргументы (допускаются любые комбинации):
  -sstring - строка сигнатур
  -t       - определять время
  -x       - использовать тестовый сценарий
  filename - файл сигнатур
  (none)   - стандартный ввод для сигнатур
 */
int main (argc, argv)
int argc;
char *argv[];
{
       int i;

       if (argc > 1)
              for (i = 1; i < argc; i++)
```

```
                        if (argv[i][0] == '-' && argv[i][1] == 's')
                                MDString (argv[i] + 2);
                        else if (strcmp (argv[i], "-t") == 0)
                                MDTimeTrial ();
                        else if (strcmp (argv[i], "-x") == 0)
                                MDTestSuite ();
                        else
                                MDFile (argv[i]);
        else
                MDFilter ();

        return (0);
}

/* Создает и выводит сигнатуру. */
static void MDString (string)
char *string;
{
        MD_CTX context;
        unsigned char digest[16];
        unsigned int len = strlen (string);

        MDInit (&context);
        MDUpdate (&context, string, len);
        MDFinal (digest, &context);

        printf ("MD%d (\"%s\") = ", MD, string);
        MDPrint (digest);
        printf ("\n");
}

/* Измерение времени создания сигнатуры для TEST_BLOCK_COUNT блоков по TEST_BLOCK_LEN байтов. */
static void MDTimeTrial ()
{
        MD_CTX context;
        time_t endTime, startTime;
        unsigned char block[TEST_BLOCK_LEN], digest[16];
        unsigned int i;
        printf ("MD%d time trial. Digesting %d %d-byte blocks ...", MD,
                TEST_BLOCK_COUNT, TEST_BLOCK_LEN[1]);

        /* Инициализация блока */
        for (i = 0; i < TEST_BLOCK_LEN; i++)
                block[i] = (unsigned char)(i & 0xff);

        /* Запуск таймера */
        time (&startTime);
        /* Создание сигнатур блоков */
        MDInit (&context);
        for (i = 0; i < TEST_BLOCK_COUNT; i++)
                MDUpdate (&context, block, TEST_BLOCK_LEN);
        MDFinal (digest, &context);

        /* Остановка таймера */
        time (&endTime);

        printf (" done\n");
        printf ("Digest = ");
        MDPrint (digest);
        printf ("\nTime = %ld seconds\n", (long)(endTime-startTime));
        printf ("Speed = %ld bytes/second\n",
                (long)TEST_BLOCK_LEN * (long)TEST_BLOCK_COUNT/(endTime-startTime));
}

/* Создает сигнатуры тестовых строк и выводит результаты. */
static void MDTestSuite ()
{
        printf ("MD%d test suite:\n", MD);

        MDString ("");
        MDString ("a");
        MDString ("abc");
        MDString ("message digest");
        MDString ("abcdefghijklmnopqrstuvwxyz");
        MDString ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789");
        MDString ("1234567890123456789012345678901234567890\
                  12345678901234567890123456789012345678901234567890");
}

/* Создает и выводит сигнатуру файла. */
static void MDFile (filename)
char *filename;
{
        FILE *file;
```

[1]В оригинале параметры в этой строке ошибочно указаны в обратном порядке. *Прим. перев.*

```
        MD_CTX context;
        int len;
        unsigned char buffer[1024], digest[16];

        if ((file = fopen (filename, "rb")) == NULL)
                printf ("%s can't be opened\n", filename);
        else {
                MDInit (&context);
                while (len = fread (buffer, 1, 1024, file))
                        MDUpdate (&context, buffer, len);
                MDFinal (digest, &context);
                fclose (file);

                printf ("MD%d (%s) = ", MD, filename);
                MDPrint (digest);
                printf ("\n");
        }
}
/* Создает сигнатуру данных со стандартного ввода и выводит результат. */
static void MDFilter ()
{
        MD_CTX context;
        int len;
        unsigned char buffer[16], digest[16];

        MDInit (&context);
        while (len = fread (buffer, 1, 16, stdin))
                MDUpdate (&context, buffer, len);
        MDFinal (digest, &context);

        MDPrint (digest);
        printf ("\n");
}

/* Выводит цифровую подпись в шестнадцатеричном формате. */
static void MDPrint (digest)
unsigned char digest[16];
{
        unsigned int i;

        for (i = 0; i < 16; i++)
                printf ("%02x", digest[i]);
}
/* Создает сигнатуру данных со стандартного ввода и выводит результат. */
static void MDFilter ()
{
        MD_CTX context;
        int len;
        unsigned char buffer[16], digest[16];

        MDInit (&context);
        while (len = fread (buffer, 1, 16, stdin))
                MDUpdate (&context, buffer, len);
        MDFinal (digest, &context);

        MDPrint (digest);
        printf ("\n");
}

/* Выводит цифровую подпись в шестнадцатеричном формате. */
static void MDPrint (digest)
unsigned char digest[16];
{
        unsigned int i;

        for (i = 0; i < 16; i++)
                printf ("%02x", digest[i]);
}
```

A.5 Тестовый набор

Тестовый набор MD4 (опция -x в командной строке драйвера) должен давать следующие результаты для MD5:

```
MD4 test suite:
MD4 ("") = 31d6cfe0d16ae931b73c59d7e0c089c0
MD4 ("a") = bde52cb31de33e46245e05fbdbd6fb24
MD4 ("abc") = a448017aaf21d8525fc10ae87aa6729d
MD4 ("message digest") = d9130a8164549fe818874806e1c7014b
MD4 ("abcdefghijklmnopqrstuvwxyz") = d79e1c308aa5bbcdeea8ed63df412da9
MD4 ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789") =
     043f8582f241db351ce627e153e7f0e4
MD4 ("12345678901234567890123456789012345678901234567890123456789012345678901234567890") =
     e33b4ddc9c38f2199c3e7b164fcc0536
```

Вопросы безопасности

Уровень безопасности рассмотренного в документе алгоритма считается достаточным для реализации гибридных систем создания цифровых подписей с очень высокой защищенностью на основе MD4 и криптосистем с открытыми ключами. Нам не известно причин, по которым алгоритм MD4 не подходил бы для схем с цифровыми сигнатурами, обеспечивающих очень высокий уровень надежности, но, поскольку алгоритм MD4 разрабатывался с основным упором на скорость, он может оказаться «на краю допустимого» с точки зрения устойчивости к криптоаналитическим атакам. После дальнейшего критического анализа может оказаться разумным принятие MD4 для приложений с очень высоким уровнем безопасности. Для использования в приложениях с очень высоким уровнем безопасности до завершения такого анализа рекомендуется использовать алгоритм MD5 [4].

Адрес автора

Ronald L. Rivest

Massachusetts Institute of Technology

Laboratory for Computer Science

NE43-324

545 Technology Square

Cambridge, MA 02139-1986

Phone: (617) 253-5880

EMail: rivest@theory.lcs.mit.edu

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru